Method and Apparatus for Pausing Execution in a Processor or the Like

Field of the Invention

The present invention pertains to a method and apparatus for pausing execution in a processor or the like. More particularly, an embodiment of the present invention pertains to controlling the pausing of execution of one of a plurality of threads so as to give preference to another of the threads or to save power.

Background of the Invention

As is known in the art, a processor includes a variety of sub-modules, each adapted to carry out specific tasks. In one known processor, these sub-modules include the following: an instruction cache, an instruction fetch unit for fetching appropriate instructions from the instruction cache; decode logic that decodes the instruction into a final or intermediate format, microoperation logic that converts intermediate instructions into a final format for execution; and an execution unit that executes final format instructions (either from the decode logic in some examples or from the microoperation logic in others).

Under operation of a clock, the execution unit of the processor system executes

successive instructions that are presented to it. As is known in the art, an instruction may be provided to the execution unit which results in no significant task performance for the processor system. For example, in the Intel® X86 processor systems, a NOP (No Operation) instruction causes the execution unit to take no action for an "instruction cycle." An instruction cycle as used herein is a set number of processor clock cycles that are needed for the processor to execute an instruction. In effect, the NOP instruction stalls the processor for one instruction cycle.

A limitation of the NOP instruction is that it stalls the processor for a set unit of time. Thus, using one or more NOP instructions, the processor can only be stalled for an amount of time equal to a whole number multiple of instruction cycles.

Another limitation of the NOP instruction is that the execution unit of the processor is unable to perform any other instruction execution. For example, instructions to be executed by the execution unit may be divided into two or more "threads." Each thread is a set of instructions to achieve a given task. Thus, if one of the threads includes a NOP instruction, this instruction is executed by the execution unit and stalls the entire processor (i.e., execution of the other thread cannot be done during the execution of the NOP instruction).

In view of the above, there is a need for an improved method and apparatus for pausing processor execution that avoids these limitations.

Summary of the Invention

According to an embodiment of the present invention, a method of pausing execution of instructions in a thread is presented. First it is determined if a next instruction for a first thread is an instruction of a first type. If it is then instruction of the first thread are prevented from being

processed for execution while instruction from a second thread can be processed for execution.

Brief Descriptions of the Drawings

Fig. 1 is a block diagram of a portion of a processor employing an embodiment of the present invention.

Fig. 2 is a flow diagram showing an embodiment of a method according to an embodiment of the present invention.

Fig. 3 is a block diagram of a portion of a processor employing an additional embodiment of the present invention.

Fig. 4 is a flow diagram showing an additional embodiment of a method according to an embodiment of the present invention.

Detailed Description

Referring to Fig. 1, an example of a portion of a processor system 10 employing an embodiment of the present invention is shown. In this embodiment, the processor is a multi-threaded processor where the execution is theoretically divided into two or more logical processors. As used herein, the term "thread" refers to an instruction code sequence. For example, in a video phone application, the processor may be called upon to execute code to handle video image data as well as audio data. There may be separate code sequences whose execution is designed to handle each of these data types. Thus, a first thread may include instructions for video image data processing and a second thread may be instructions for audio data processing. In this example, there is a single execution unit (out of order execution unit 31),

which may execute one instruction at a time. The processor system 10, however, may be treated as two logical processors, a first logical processor executing instructions from the first thread (Thread 0) and a second logical processor executing instructions from the second thread (Thread 1).

In this embodiment of the processor system 10, instructions are fetched by a fetch unit 11 and supplied to a queue 13 and stored as part of the thread 0 queue or the thread 1 queue. One skilled in the art will appreciate that the queues used in processor system 10 may be used to store more than two threads. Instructions from the two threads are supplied to a mulitplexer (MUX) 15, and control logic 17 is used to control whether instructions from thread 0 or thread 1 are supplied to a decode unit 21. Decode unit 21 may convert an instruction into two or more microinstructions and supplies the instructions to queue 23. The outputs of queue 23 are supplied to a MUX which supplies instruction from thread 0 or thread 1 to a rename/allocation unit 27 based on operation of control logic 26. The rename/allocation unit 27, in turn, supplies instructions to queue 28. MUX 29 selects between the thread 0 queue and the thread 1 queue based on the operation of schedule control logic 30, which also receives the same inputs as MUX 29. The output of MUX 29 is supplied to an execution unit 31 which executes the instruction. The instruction is then placed in queue 33. The outputs of queue 33 are supplied to a MUX 34 which sends instruction from thread 0 and thread 1 to a retire unit 36 based on the operation of control logic 35.

According to a first embodiment of the present invention, a pause instruction is used to suspend processing of instructions from a thread. In Fig. 1, the pause instruction is fetched by fetch unit 11 and stored in the thread 0 queue, in this example. The output of the thread 0 queue

is supplied via MUX 15 to decode unit 21 which decodes the pause instructior into two microinstructions: a SET instruction and a READ instruction. At the decode unit 21, a SET instruction causes a value to be stored in memory (e.g., a bit flag 19) indicating that a SET instruction has been received for a particular thread (thread 0 in this example). The SET instruction is then fed into the "pipeline" which includes rename/allocation unit 27 and execution unit 31 and the associated queues in this embodiment. Execution unit 31 takes no action on the SET instruction (i.e., treats it as the known NOP instruction). Once the SET instruction is retired by retire unit 26, the flag 19 is reset.

The READ instruction at decode unit 21 is not placed into the pipeline until the flag 19 for that flag is reset. Accordingly, if there are instructions from thread 1 in queue 13, these instructions can be decoded by decode unit 21 and placed into the pipeline. Thus, depending on the number of thread 1 instructions in queues 23, 28 and 33, will affect how long the execution of thread 0 is paused (i.e., the greater number of thread 1 instructions in the pipeline, the longer it will take the SET instruction to reach retire unit 36). Once the flag 19 is reset, the READ instruction is sent to queue 23 and is eventually sent to execution unit 31. As with the SET instruction, execution unit takes not action as with a NOP instruction. In this embodiment of the present invention, decode unit 21 alternates decoding of instructions from thread 0 and thread 1. After a SET instruction for thread 0, for example, the decode alternates between decoding instructions from thread 1 and checking the value of flag 19 until it is reset.

An example of the operation of decode unit 21 in this embodiment is shown in Fig. 2. After decoding, in block 40, the instruction from the next thread is determined. In decision block 41, it is determined whether the instruction is a SET instruction. If it is, then control passes to

block 43 where the bit flag in memory is set. In block 47, the SET instruction is placed into the pipeline for the execution unit. Control then returns to block 40 to determine the next instruction from the next thread. If the instruction is not a SET instruction, control passes to decision block 45 to determine if the instruction is a READ instruction. If it is, then control passes to decision block 49 to determine if the appropriate bit flag in memory is set. If the bit flag in memory is set, then control passes to block 51 where the instruction is held back from the pipeline (thus, temporarily blocking execution of instructions from that particular thread). Control then shifts to block 40 to determine the next instruction from the next thread. If the bit flag is not set (decision block 49), then control passes to block 53 where the instruction (in this case the READ instruction) is placed into the pipeline for execution. As stated above, the bit flag is reset in this embodiment when the SET instruction is retired. Control then returns to block 40 to determine the next instruction from the next thread. Likewise, if the instruction is neither a SET instruction nor a READ instruction, it is placed into the pipeline for execution in a normal manner.

As seen from the above, the SET instruction works to effect a pause in execution for that thread until the instruction is retired. This is because the following READ instruction is not placed into the pipeline until the SET instruction is retired effectively blocking execution of the following instructions from that thread. During the pause of one thread, instructions from that thread are prevented from being processed for execution (e.g., placed into the pipeline, sent to the execution unit, etc.) while instructions from another thread can be processed for execution. When execution of a thread is paused, overall power consumption for the processing system may be reduced.

According to another embodiment of the present invention, a pause instruction is

implemented with a timer or counter. As shown in Fig. 3, the memory flag 19 of Fig. 1 is replaced by a counter 39. As a first example, when decode unit 21determines that the next instruction from a first thread is a pause instruction (i.e., an instruction having a particular bit format), then a predetermined value is loaded into counter 39. In this example, counter 39 counts down from the predetermined value to zero. While counter 39 counts down to zero, instructions from the second thread (e.g., thread 1) are decoded and loaded into the pipeline. In this example, decode unit 21 alternates between checking the value of counter 39 (instead of decoding instructions from thread 0) and decoding instructions from thread 1. Once the counter has finished (e.g., reached zero), the next instruction from that thread can be loaded into the pipeline. As a second example, the pause instruction will include an operand (i.e., a value to be loaded into the timer). Accordingly, this allows decode unit 21 to load the operand value into counter 39 so that the length of time for the pause instruction can be set.

An example of the operation of the processing system of Fig. 3 is shown in Fig. 4. In decision block 60 it is determined if the counter has reached a predetermined value for the current thread. If no counter has been set or if the value has reached the predetermined value (e.g., zero), then control passes to block 61 to determine the next instruction for the current thread. If this instruction is a pause instruction (decision block 63), then control passes to decision block 65 to determine whether an operand is associated with the pause instruction. If an operand is associated with the pause instruction, then control passes to block 67 to load the value into the counter (control then passes to block 73 to change to the next thread). If an operand is not associated with the pause instruction, then control passes to block 65 to load a predetermined value into the counter (again control then passes to block 73 to change to the next thread). If in

decision block 63, the instruction is not a pause instruction, then control passes to block 71 to load the instruction into the pipeline.

According to an embodiment of the present invention, the use of the pause instruction can be an indication by the operating system that the processing system hardware can go into a low-power mode. Thus, execution of operating system code (or any other software code) at the processor system may cause a pause instruction to be forward to the decode unit. As described above, pausing execution of a thread may lead to a decrease in overall power consumption. In response to decoding a pause instruction, the processing system 10 may take other steps to lower overall power consumption further as desired.

Although several embodiments are specifically illustrated and described herein, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.